Source Code for U.S Patent Application -- Copyright 2000 Xippix, Inc.
Attorney Docket No.: 020133-000300US

METHOD AND SYSTEM USING NON-UNIFORM IMAGE BLOCKS
FOR RAPID INTERACTIVE VIEWING OF DIGITAL IMAGES OVER A NETWORK

Contains two C++ language source code files,

GenCookie.c
GenCookie.h

MICROFICHE APPENDIX

```
// ==========================================================================
 // File: GenCookie.c
 // Copyright, 1999, 2000, Xippix, Inc.
 // The terminology "cookie" is because the pattern of image blocks selected
 // looks, when graphed, like the pattern of a rectangular cookie cutter
 // stamped in an overlapping manner on dough.)
 // ==========================================================================
 // constants from TPCApp.h
 #include <CMApplication.h>

 // ==========================================================================
 // Resource-dependent constants
 // If you port these to another application, you have to make sure all these
 // are set up in the resource file.  E.g., set up window 1450 (WIND_Tonal)
 // for the tonal budgeting window.
 // ==========================================================================
 // constants from TPCApp.h
 #include "LApplication.h" // You have to call this before you declare const.

 #include "FontRs.h"
 const ResIDT WIND_Tonal        =  1450; // TpcRs_h
 const ResIDT WIND_picker  = 10800; // ViewMsg_h
 const ResIDT WIND_pickerS = 10801;
 const ResIDT WIND_Block   = 11100; // TpcRs_h
 #include "GeneralData.h"
 #include "GenPreview.h"
 #include "GenPreview.cmd"  // Defines of messages from GenPreview, form
"msg_{foobar}"
 #include "HLS.h"
 #include "ViewBigCube.h"

 #include "NumScrollBox.h"
 #include "DblScrollBox.h"

 #include "GenCookie.h"
 #include "Tonal.h"

 #include "JBfileUtils.h"
 #include "busy.h"
 // CW10 Version had: #include <String_Utils.h> // for c2pstr

 #include "tpu_pane.h"
 #include "tpu_alert.h"
 #include "tpu_number.h"
 #include "tpu_trig.h"


 #include "AGA.h"
 #include "AGAPP.h"

 #include <Quickdraw.h>

 #include <math.h> // for "pow"

 // TL file system
 #include "TLTypedefs.h" // Has typedefs "s8", etc., needed by TLFileSys.h
 #include "TLFileSys.h"
```

```
// Following are for the interface to the application data
#include <stdlib.h>

static int hasdata = 0;

#ifdef MEM_DEBUG
#include "smrtheap.hpp"
// #include "shmac.c"
#endif

static int staticbreak = 0;
static unsigned char staticchar = 0;
static unsigned char flipping = 1;

// Block Stuff
// Following leftover from TPCViewPreview.c
static int wwidth  = 510; // use 0 to 499 incl. 320; // 460
static int wheight = 570; // use 0 to 559 incl. 460;
static int finalx = 449;
static int finaly = 499;
static int panw = 96; // 12 * 8bits;
static int panh = 96; // 96 rows thereof
static unsigned char DoDrawBlock = 0;

static unsigned char everdrawn = 0;
static unsigned char newway = 1;

static void ToggleDoDraw()
{
     if (!DoDrawBlock)
          DoDrawBlock = 1;
     else
          DoDrawBlock = 0;
}


static RGBColor gray  = {0xc000, 0xc000, 0xc000};
static RGBColor dark  = {0x9000, 0x9000, 0x9000};
static RGBColor black = {0x0000, 0x0000, 0x0000};
static RGBColor white = {0xffff, 0xffff, 0xffff};


// GenCookie.h

////////////////////////////////////////////////////////////////////////////////
/
 //
 // Start class ViewCookie
 //
 // Cooked
 /* Proc */ ViewCookie::ViewCookie
                          (
                             const SPaneInfo &tpi,
                             const SViewInfo &tvi,
                             unsigned short yoffset, // GenTonal.h
                             WinCookie   *dlgbox,
```

```
                                LCommander  *inSuperCommander,
                                GeneralData *gen
                              )
                                              : LView (tpi, tvi)
{
     capp__ = inSuperCommander;              // Untyped ptr to App.
     gen__  = gen;
     dlg__  = dlgbox;

     secondPort = 0;

     Rect frame;
     CalcLocalFrameRect(frame);
     gframe = frame; // Block;

     acount = lcount = 0; // Block;

     stackcount = 0;

     image.tl.x = 0;
     image.tl.y = 0;
     image.br.x = finalx;
     image.br.y = finaly;
     image.id = 100;
     image.status = 0;// 0 if no image data for the block; 1 if here; 2 if on
order
     image.next = 0;  // we don't care. It's a singleton.

     first = -1;

     blonkall.left  = 0;
     blonkall.right = finalx +1; // PaintRect covers its top & left parms,
     blonkall.top = 0;
     blonkall.bottom = finaly + 1;

     // Next action at ClickSelf; then AdjustCursorSelf
}


// Cooked
unsigned char ViewCookie::DoBlocksIntersect (uRect *a, uRect *b, unsigned char
DoCalc)
{
     if ( (a->tl.x <= b->br.x) && (b->tl.x <= a->br.x) &&
          (a->tl.y <= b->br.y) && (b->tl.y <= a->br.y) )
     {
           if (DoCalc)
           {
                 if (a->tl.x > b->tl.x) intercept.tl.x = a->tl.x; else
intercept.tl.x = b->tl.x;
                 if (a->tl.y > b->tl.y) intercept.tl.y = a->tl.y; else
intercept.tl.y = b->tl.y;
                 if (a->br.x < b->br.x) intercept.br.x = a->br.x; else
intercept.br.x = b->br.x;
                 if (a->br.y < b->br.y) intercept.br.y = a->br.y; else
intercept.br.y = b->br.y;
           }
```

```
            return (1);
        }
        else
            return (0);
}


// Find first intersecting block.
// Cooked
int ViewCookie::FindBlockIntersectingBlock (uRect *target)
{
    int i; // Block
    unsigned char finished = 0;
    i = first;

    if (first < 0)
        return (-1); // Failure

    // 0-terminated linked list.
    while (!finished)
    {
        if (DoBlocksIntersect (target, &rlist[i], 1))
            return (i);
        i = rlist[i].next;
        if (i < 0)
            finished = 1;
    }
    return (-1); // Failure.
}


void ViewCookie::ReceiveOrderedBoxes ()
{
    int i; // Block
    unsigned char finished = 0;
    i = first;

    // 0-terminated linked list.
    while (!finished)
    {
        if (rlist[i].status == 2)
            rlist[i].status = 1;
        i = rlist[i].next;
        if (i < 0)
            finished = 1;
    }
}


int ViewCookie::FindPredecessor (int block)
{
    int i, j; // Block
    unsigned char finished = 0;
    i = first;

    // 0-terminated linked list.
    while (!finished)
```

```
        {
                j = rlist[i].next;
                if (j == block)
                        return (i);
                i = j;
                if (j < 0)
                        finished = 1;
        }
        return (-1); // Failure.
}


void ViewCookie::DelRectangle (int block)
{
        int predecessor;

        if (block != first)
        // Stitch linked list back together
        {
                predecessor = FindPredecessor (block);
                        rlist[predecessor].next = rlist[block].next;
        }
        else
        // if (block == first)
                first = rlist[block].next;

        lcount--;
}


// GenBlock.h
void ViewCookie::AddRectangle (int minx, int miny, int maxx, int maxy, int
status)
{
        // Set rectangle 1
        rlist[acount].tl.x = minx;
        rlist[acount].tl.y = miny;
        rlist[acount].br.x = maxx;
        rlist[acount].br.y = maxy;
        rlist[acount].id = acount;
        rlist[acount].status = status;
        rlist[acount].next = first; // Temp. Add as new first in list. The list is
(-1)-terminated.

        first = acount;                 // Temp.
        acount++;
        lcount++;
        // SewInLists(); // Later: Keep four linked list of next pointers in the
order of:
        // tl.x, tl.y, br.x, br.y. This will speed up searches.
(FindBlockIntersectingBlock, etc.)
}


static unsigned char PutABreakpointHere()
{
        staticbreak++;
```

```
        return (staticbreak);
}


static unsigned char PutABreakpointThere()
{
        unsigned char fazz = PutABreakpointHere();
        return (fazz);
}


// Cooked
unsigned char ViewCookie::ItsTall (uRect *r)
{
        if ((r->br.y - r->tl.y) > (r->br.x - r->tl.x))
                return (true);
        else
                return (false);
}


// Cooked
void ViewCookie::FactorBlock(uRect *box)
{
        int i, j;
        int countincluded;
        unsigned char tlin, trin, blin, brin; // top left in, etc
        unsigned char ht, hb, vl, vr; // horizontal top, etc.  for 0 intercepts
case.
        unsigned char tl, bl, tr, br; // top left, etc.        for 1 intercept
case.
        unsigned char ls, rs, ts, bs; // left side, etc.       for 2 intercept
case.
     unsigned char fazz; // Debug.

        if (!acount)
                fazz = PutABreakpointHere();

     int minxi, maxxi, minyi, maxyi;

        int blockminxi = box->tl.x; // Break these out for easier parsing.
        int blockmaxxi = box->br.x;
        int blockminyi = box->tl.y;
        int blockmaxyi = box->br.y;

        minxi = intercept.tl.x; // Break these out for easier parsing.
        maxxi = intercept.br.x;
        minyi = intercept.tl.y;
        maxyi = intercept.br.y;

        // To HERE

        // Get the count of included points.
        countincluded = 0;
        tlin = trin = blin = brin = 0;
```

```
        /*
        WARNING!! It took me a long time to get the degree-of-inclusiveness
        (the choice of "<" vs "<=") right in the following. The solution turns
        out to be that a point is in if it leaves white space on the continuation
        of both of its joining edges. E.g., the bottom right point is in if it
        leaves white space at the bottom (maxyi < blockmaxyi) and at the right
        (maxxi < blockmaxxi). As far as the remaining two inequalities goes,
        they can be satisfied weakly: (maxxi >= blockminxi; maxyi >= blockminyi).
        */

        // Top left
        // Must leave white space at left (blockminxi) and top (blockminyi).
        if (minxi > blockminxi && minxi <= blockmaxxi &&
            minyi > blockminyi && minyi <= blockmaxyi) // WARNING!! See note above
        {
              countincluded++;
              tlin = 1;
        }


        // Top right
        // Must leave white space at right (blockmaxxi) and top (blockminyi).
        if (maxxi >= blockminxi && maxxi <  blockmaxxi &&
            minyi >  blockminyi && minyi <= blockmaxyi) // WARNING!! See note
above
        {
              countincluded++;
              trin = 1;
        }


        // Bottom left
        // Must leave white space at left (blockminxi) and bottom (blockaxyi).
        if (minxi >  blockminxi && minxi <= blockmaxxi &&
            maxyi >= blockminyi && maxyi <  blockmaxyi) // WARNING!! See note
above
        {
              countincluded++;
              blin = 1;

        }


        // Bottom right
        // Must leave white space at right (blockmaxxi) and bottom (blockmaxyi).

        if (maxxi >= blockminxi && maxxi < blockmaxxi &&
            maxyi >= blockminyi && maxyi < blockmaxyi) // WARNING!! See note above
        {
              countincluded++;
              brin = 1;

        }


        switch (countincluded)
        {
```

```
            default:
            case 0:
                  // We will make 2 new ones or 3 new ones
                  // Watch out for the difference between "<" and "<="; the
                  // question is always: Is white space left.
                  vl = vr = ht = hb = 0;
                  if (maxxi < blockmaxxi && maxxi >= blockminxi)
                        vl = 1;
                  if (minxi > blockminxi && minxi <= blockmaxxi)
                        vr = 1;
                  if (maxyi < blockmaxyi && maxyi >= blockminyi)
                        ht = 1;
                  if (minyi > blockminyi && minyi <= blockmaxyi)
                        hb = 1;

      ////////////////////////////////////////////////////////////////////////
/
                  //
                  // First vertical-region cases
                  if (vl && !vr)
                  {
                  //
                  //  -------------
                  // |.1.| 2       |
                  // |...|         |  Region 1 is the view
                  // |...|         |
                  //  -------------
                  IntersectBox4 (maxxi+1,    blockminyi, blockmaxxi,
blockmaxyi); // Rect 2
                  }
                  else
                  if (vr && !vl)
                  {
                  //
                  //  -------------
                  // | 1 |.2.......|
                  // |   |.........|  Region 2 is the view
                  // |   |.........|
                  //  -------------
                  IntersectBox4 (blockminxi, blockminyi, minxi-1,
blockmaxyi); // Rect 1
                  }
                  else
                  if (vr && vl)
                  {
                  //
                  //  -----------------
                  // | 1 |.2.......|3  |
                  // |   |.........|   |  Region 2 is the view
                  // |   |.........|   |
                  //  -----------------
                  IntersectBox4 (blockminxi, blockminyi, minxi-1,
blockmaxyi); // Rect 1
                  IntersectBox4 (maxxi+1,    blockminyi, blockmaxxi,
blockmaxyi); // Rect 3
                  }
```

```
//////////////////////////////////////////////////////////////////////////
/
                    //
                    // Then horizontal-region cases
                    else
                    if (ht && !hb)
                    {
                    //
                    //  -------------
                    // |.1...........|
                    // |-------------|  Region 1 is the view
                    // | 2           |
                    //  -------------
                    IntersectBox4 (blockminxi, maxyi+1,    blockmaxxi,
blockmaxyi); // Rect 2
                    }
                    else
                    if (hb && !ht)
                    {
                    //
                    //  -------------
                    // | 1           |
                    // |-------------|  Region 2 is the view
                    // |.2...........|
                    //  -------------
                    IntersectBox4 (blockminxi, blockminyi, blockmaxxi, minyi-1);
// Rect 1
                    }
                    else
                    if (hb && ht)
                    {
                    //
                    //  -------------
                    // | 1           |
                    // |-------------|  Region 2 is the view
                    // |.2...........|
                    // |-------------|
                    // | 3           |
                    //  -------------
                    IntersectBox4 (blockminxi, blockminyi, blockmaxxi, minyi-1);
// Rect 1
                    IntersectBox4 (blockminxi, maxyi+1,    blockmaxxi,
blockmaxyi); // Rect 3
                    }
                    break;

              case 1:
                    // We will make 3 new ones
                    tl = tr = bl = br = 0;
                    if (maxxi < blockmaxxi && maxyi < blockmaxyi)
                          tl = 1;
                    else
                    if (maxxi < blockmaxxi && minyi > blockminyi)
                        bl = 1;
                    else
                    if (minxi > blockminxi && maxyi < blockmaxyi)
```

```
                        tr = 1;
                  else
                  if (minxi > blockminxi && minyi > blockminyi)
                        br = 1;

                  if (tl)
                  {
                  if (!flipping || ItsTall(box))
                  {
                  // Tall
                  //  -----------------
                  // |.1.........| 2   |
                  // |...........|     |
                  // |...........|     |
                  // |...........|     |
                  // |-----------------|
                  // | 3               |
                  // |                 |
                  // |                 |
                  // |                 |
                  //  -----------------
                  //
                  IntersectBox4 (maxxi+1,    blockminyi, blockmaxxi, maxyi);
// Rect 2
                  IntersectBox4 (blockminxi, maxyi+1,    blockmaxxi,
blockmaxyi); // Rect 3
                  } // if (tl) tall
                  else
                  {
                  // Wide
                  //  -----------------
                  // |.1.........| 3   |
                  // |...........|     |
                  // |...........|     |
                  // |...........|     |
                  // |-----------|     |
                  // | 2         |     |
                  // |           |     |
                  //  -----------------
                  //
                  IntersectBox4 (blockminxi, maxyi+1,    maxxi,
blockmaxyi); // Rect 2
                  IntersectBox4 (maxxi+1,    blockminyi, blockmaxxi,
blockmaxyi); // Rect 3
                  } // if (tl) wide
                  } // if (tl)

                  if (bl)
                  {
                  if (!flipping || ItsTall(box))
                  {
                  // Tall
                  //  -----------------
                  // | 1               |
                  // |                 |
                  // |                 |
                  // |                 |
```

```
                    // |-----------------|
                    // |.2.........| 3    |
                    // |..........|      |
                    // |..........|      |
                    // |..........|      |
                    //  -----------------
                    //
                    IntersectBox4 (blockminxi, blockminyi, blockmaxxi, minyi-1);
// Rect 1
                    IntersectBox4 (maxxi+1,    minyi,      blockmaxxi,
blockmaxyi); // Rect 3
                    } // if (bl) tall
                    else
                    {
                    // Wide
                    //  -----------------
                    // | 1          | 3   |
                    // |            |     |
                    // |            |     |
                    // |            |     |
                    // |------------|     |
                    // |.2.........|     |
                    // |..........|     |
                    //  -----------------
                    //
                    IntersectBox4 (blockminxi, blockminyi, maxxi,      minyi-1);
// Rect 1
                    IntersectBox4 (maxxi+1,    blockminyi, blockmaxxi,
blockmaxyi); // Rect 3
                    } // if (bl) wide
                    } // if (bl)

                    if (tr)
                    {
                    if (!flipping || ItsTall(box))
                    {
                    // Tall
                    //  -----------------
                    // | 1          |.2...|
                    // |            |.....|
                    // |            |.....|
                    // |            |.....|
                    // |-----------------|
                    // | 3               |
                    // |                 |
                    // |                 |
                    // |                 |
                    //  -----------------
                    //
                    IntersectBox4 (blockminxi, blockminyi, minxi-1,    maxyi);
// Rect 1
                    IntersectBox4 (blockminxi, maxyi+1,    blockmaxxi,
blockmaxyi); // Rect 3
                    } // if (tr) tall
                    else
                    {
                    // Wide
```

```
                    //  -----------------
                    // | 1          |.2...|
                    // |            |.....|
                    // |            |.....|
                    // |            |.....|
                    // |            |-----|
                    // |            | 3   |
                    // |            |     |
                    //  -----------------
                    //
                    IntersectBox4 (blockminxi, blockminyi, minxi-1,
blockmaxyi); // Rect 1
                    IntersectBox4 (minxi,      maxyi+1,    blockmaxxi,
blockmaxyi); // Rect 3
                    } // if (tr) wide
                    } // if (tr)

                    if (br)
                    {
                    if (!flipping || ItsTall(box))
                    {
                    // Tall
                    //  -----------------
                    // | 1               |
                    // |                 |
                    // |                 |
                    // |                 |
                    // |-----------------|
                    // | 2          |.3...|
                    // |            |.....|
                    // |            |.....|
                    // |            |.....|
                    //  -----------------
                    //
                    IntersectBox4 (blockminxi, blockminyi, blockmaxxi, minyi-1);
// Rect 1
                    IntersectBox4 (minxi,      minyi,      blockmaxxi,
blockmaxyi); // Rect 3
                    } // if (tr) tall
                    else
                    {
                    // Wide
                    //  -----------------
                    // | 1          | 2   |
                    // |            |     |
                    // |            |     |
                    // |            |     |
                    // |            |-----|
                    // |            |.3...|
                    // |            |.....|
                    //  -----------------
                    //
                    IntersectBox4 (blockminxi, blockminyi, minxi-1,
blockmaxyi); // Rect 1
                    IntersectBox4 (minxi,      blockminyi, blockmaxxi, minyi-1);
// Rect 2
                    } // if (tr) wide
```

```
                        } // if (tr)

                        break;

                case 2:
                        // We will make 4 new ones
                        ls = rs = ts = bs = 0;
                        if (tlin && blin)
                              rs = 1;
                        else
                        if (trin && brin)
                              ls = 1;
                        else
                        if (tlin && trin)
                              bs = 1;
                        else
                        if (blin && brin)
                              ts = 1;

                        if (ts)
                        {
                        if (!flipping || ItsTall(box))
                        {
                        // Tall
                        //
                        //  -----------------
                        // | 1 |.2.....| 3   |
                        // |   |.......|     |
                        // |-----------------|
                        // | 4               |
                        // |                 |
                        // |                 |
                        // |                 |
                        // |                 |
                        // |                 |
                        //  -----------------
                        //
                        IntersectBox4 (blockminxi, blockminyi, minxi-1,    maxyi);
// Rect 1
                        IntersectBox4 (maxxi+1,    blockminyi, blockmaxxi, maxyi);
// Rect 3
                        IntersectBox4 (blockminxi, maxyi+1,    blockmaxxi,
blockmaxyi); // Rect 4
                        } // End if (ts) tall
                        else
                        {
                        // Wide
                        //
                        //  -----------------
                        // | 1 |.2.....| 3   |
                        // |   |.......|     |
                        // |   |-------|     |
                        // |   | 4     |     |
                        // |   |       |     |
                        // |   |       |     |
                        //  -----------------
                        //
```

```
                    IntersectBox4 (blockminxi, blockminyi, minxi-1,
blockmaxyi); // Rect 1
                    IntersectBox4 (maxxi+1,    blockminyi, blockmaxxi,
blockmaxyi); // Rect 3
                    IntersectBox4 (minxi,      maxyi+1,    maxxi,
blockmaxyi); // Rect 4
                    } // End if (ts) wide
                    } // End if (ts)

                    if (bs)
                    {
                    if (!flipping || ItsTall(box))
                    {
                    // Tall
                    //
                    //  -----------------
                    // | 1               |
                    // |                 |
                    // |                 |
                    // |                 |
                    // |                 |
                    // |                 |
                    // |-----------------|
                    // | 2 |.3.....| 4   |
                    // |   |.......|     |
                    //  -----------------
                    IntersectBox4 (blockminxi, blockminyi, blockmaxxi, minyi-1);
// Rect 1
                    IntersectBox4 (blockminxi, minyi,      minxi-1,    maxyi);
// Rect 2
                    IntersectBox4 (maxxi+1,    minyi,      blockmaxxi, maxyi);
// Rect 4
                    } // End if (bs) tall
                    else
                    {
                    // Wide
                    //
                    //  -----------------
                    // | 1 | 2     | 3   |
                    // |   |       |     |
                    // |   |-------|     |
                    // |   |.4.....|     |
                    // |   |.......|     |
                    // |   |.......|     |
                    //  -----------------
                    //
                    IntersectBox4 (blockminxi, blockminyi, minxi-1,
blockmaxyi); // Rect 1
                    IntersectBox4 (minxi,      blockminyi, maxxi,      minyi-1);
// Rect 2
                    IntersectBox4 (maxxi+1,    blockminyi, blockmaxxi,
blockmaxyi); // Rect 3
                    } // End if (bs) wide
                    } // End if (bs)

                    if (ls)
                    {
```

```
                    if (!flipping || ItsTall(box))
                    {
                    // Tall
                    //
                    //  -----------------
                    // | 1               |
                    // |                 |
                    // |-----------------|
                    // | 2.| 4           |
                    // |...|             |
                    // |-----------------|
                    // | 3               |
                    // |                 |
                    // |                 |
                    // |                 |
                    // |                 |
                    //  -----------------
                    //
                    IntersectBox4 (blockminxi, blockminyi, blockmaxxi, minyi-1);
// Rect 1
                    IntersectBox4 (blockminxi, maxyi+1,    blockmaxxi,
blockmaxyi); // Rect 3
                    IntersectBox4 (maxxi+1,    minyi,      blockmaxxi, maxyi);
// Rect 4
                    } // End if (ls) tall
                    else
                    {
                    // Wide
                    //
                    //  -----------------
                    // | 1 | 4           |
                    // |   |             |
                    // |---|             |
                    // |.2.|             |
                    // |...|             |
                    // |---|             |
                    // | 3 |             |
                    //  -----------------
                    //
                    IntersectBox4 (blockminxi, blockminyi, maxxi,      minyi-1);
// Rect 1
                    IntersectBox4 (blockminxi, maxyi+1,    maxxi,
blockmaxyi); // Rect 3
                    IntersectBox4 (maxxi+1,    blockminyi, blockmaxxi,
blockmaxyi); // Rect 4
                    } // End if (ls) wide
                    } // End if (ls)

                    if (rs)
                    {
                    if (!flipping || ItsTall(box))
                    {
                    // Tall
                    //
                    //  -----------------
                    // | 1               |
                    // |                 |
```

```
                    //  |-----------------|
                    //  |  2  |.4.........|
                    //  |     |..........|
                    //  |-----------------|
                    //  |  3              |
                    //  |                 |
                    //  |                 |
                    //  |                 |
                    //  |                 |
                    //   -----------------
                    //
                    IntersectBox4 (blockminxi, blockminyi, blockmaxxi, minyi-1);
// Rect 1
                    IntersectBox4 (blockminxi, minyi,      minxi-1,    maxyi);
// Rect 2
                    IntersectBox4 (blockminxi, maxyi+1,    blockmaxxi,
blockmaxyi); // Rect 3
                    } // End if (rs) tall
                    else
                    {
                    // Wide
                    //
                    //   --------------
                    //  | 1          | 2 |
                    //  |            |   |
                    //  |            |---|
                    //  |            |.3.|
                    //  |            |...|
                    //  |            |---|
                    //  |            | 4 |
                    //   --------------
                    //
                    IntersectBox4 (blockminxi, blockminyi, minxi-1,
blockmaxyi); // Rect 1
                    IntersectBox4 (minxi,      blockminyi, blockmaxxi, minyi-1);
// Rect 2
                    IntersectBox4 (minxi,      maxyi+1,    blockmaxxi,
blockmaxyi); // Rect 4
                    } // End if (rs) wide
                    } // End if (rs)

                    break;

               case 4:
                    // We will make 5 new ones
                    //
                    if (!flipping || ItsTall(box))
                    {
                    // Tall
                    //   -----------------
                    //  | 1               |
                    //  |-----------------|
                    //  | 2  |.3.....| 4  |
                    //  |    |.......|    |
                    //  |-----------------|
                    //  | 5               |
                    //  |                 |
```

```
                    // |                 |
                    // |                 |
                    //  -----------------
                    //
                    IntersectBox4 (blockminxi, blockminyi, blockmaxxi, minyi-1);
// Rect 1
                    IntersectBox4 (blockminxi, minyi,      minxi-1,    maxyi);
// Rect 2
                    IntersectBox4 (maxxi+1,    minyi,      blockmaxxi, maxyi);
// Rect 4
                    IntersectBox4 (blockminxi, maxyi+1,    blockmaxxi,
blockmaxyi); // Rect 5
                    }
                    else
                    // if ((blockmaxyi - blockminyi) <= (blockmaxxi - blockminxi))
                    {
                    // Wide
                    //  -----------------
                    // | 1 | 2     | 5   |
                    // |   |-------|     |
                    // |   |.3.....|     |
                    // |   |.......|     |
                    // |   |-------|     |
                    // |   | 4     |     |
                    // |   |       |     |
                    //  -----------------
                    //
                    IntersectBox4 (blockminxi, blockminyi, minxi-1,
blockmaxyi); // Rect 1
                    IntersectBox4 (minxi,      blockminyi, maxxi,      minyi-1);
// Rect 2
                    IntersectBox4 (minxi,      maxyi+1,    maxxi,
blockmaxyi); // Rect 4
                    IntersectBox4 (maxxi+1,    blockminyi, blockmaxxi,
blockmaxyi); // Rect 5
                    }
                    break;

      } // End switch (countincluded)
 }


 void ViewCookie::IntersectBox4 (int minx, int miny, int maxx, int maxy)
 {
      uRect box;
      box.tl.x = minx;
      box.tl.y = miny;
      box.br.x = maxx;
      box.br.y = maxy;

      IntersectBox1 (&box);
 }


 // Cooked
 // GenCookie.h
 void ViewCookie::IntersectBox1(uRect *box)
```

```
{
	int block;

	if (box->br.x > finalx)
		box->br.x = finalx; // Clip to image

	if (box->br.y > finaly)
		box->br.y = finaly; // Clip to image

	block = FindBlockIntersectingBlock (box);

	if (block < 0) // No intersecting block found.
	{
		AddRectangle (box->tl.x, box->tl.y, box->br.x, box->br.y, 2); // Not
yet ready.
		return;
	}
	else
		FactorBlock (box);
}


// Block
static void BlockLegend (int xval, int yval, int id, unsigned char *sid)
{
	sprintf ((char *)sid, "%d", id);

	::TextFont (geneva);
	::TextSize (10);
	::MoveTo (xval+8, yval+12);
	::DrawString (c2pstr((char *)sid));
}


// Block
void ViewCookie::RenderRect (uRect *r)
// Box around the whole frame, drawn on the first-last row-col of the frame
{
	unsigned char sid[4];
	Rect blonk;

	//////////////////////////////////////////////////////////
	//
	// First Block out entire underlying area
	//
	if (!r->status)
		::RGBForeColor (&white);
	else
	if (r->status == 1)
		::RGBForeColor (&gray);
	else
		::RGBForeColor (&dark);


	blonk.left   = r->tl.x;
	blonk.right  = r->br.x+1;
	blonk.top    = r->tl.y;
```

```
	blonk.bottom = r->br.y+1;
	PaintRect (&blonk);

	////////////////////////////////////////////////////////
	//
	// Then draw boundary
	::TextMode (srcOr); // MQ
	::RGBForeColor (&black);

	::MoveTo (r->tl.x, r->tl.y);
	::LineTo (r->br.x, r->tl.y);
	::LineTo (r->br.x, r->br.y);
	::LineTo (r->tl.x, r->br.y);
	::LineTo (r->tl.x, r->tl.y);

	BlockLegend (r->tl.x, r->tl.y, r->id, sid);
}


// Block
/* Proc */ void ViewCookie::DrawAllBoxes ()
{
	::PenNormal(); // MQ
	::GetPort (&secondPort); // MQ

	// First blonk out underlying area.
	::PenMode (patCopy);	// MQ
	::TextMode (srcOr); // MQ

	////////////////////////////////////////////////////////
	//
	// First Block out entire underlying area
	//
	::RGBForeColor (&white);
	PaintRect (&blonkall);
	everdrawn = 0; // Reset for rubberbanding

	::TextMode (srcOr); // MQ
	::RGBForeColor (&black);

	RenderRect (&image);

	////////////////////////////////////////////////////////
	//
	// Now do individual rectangles
	//
	int i; // Block
	unsigned char finished = 0;
	i = first;

	if (first < 0)
		return;

	// 0-terminated linked list.
	while (!finished)
	{
		RenderRect (&rlist[i]);
```

```
            i = rlist[i].next;
            if (i < 0)
                  finished = 1;
      }
}


/* Proc */ void ViewCookie::DrawSelf()
{
      DrawAllBoxes ();
}


// Basic Mouse IO. 1 of 5.
// Cooked.
/* Proc */ void ViewCookie::ClickSelf (const SMouseDownEvent &inMouseDown)
{
      unsigned short Sep = 0; // Default; may be overridden

      ToggleDoDraw ();
}


// Basic Mouse IO. 2 of 5.
// Raw
/* Proc */ void ViewCookie::EventMouseUp (const EventRecord &inMacEvent)
{
      uRect box;

      box.tl.x = xp;
      box.tl.y = yp;
      box.br.x = xp + panw - 1;
      box.br.y = yp + panh - 1;
      box.id = 0;
      box.status = 0;
      box.next = 0; // We don't care; it's a singleton.

      ToggleDoDraw();
      IntersectBox1(&box);
      DrawAllBoxes();
      ReceiveOrderedBoxes();
}


// GenTonal.h
/* Proc */ void ViewCookie::DrawDotBox ()
{
      if (!secondPort)
            return;

      everdrawn = 1;

      ::SetPort (secondPort);    // MQ Test this 980112

      ::PenNormal(); // MQ
      ::RGBForeColor (&black);
```

```
    Point varpt;

    ::PenMode (patXor);     // MQ
    ::PenPat (&qd.gray);

    ::MoveTo (view.tl.x, view.tl.y);
    ::LineTo (view.br.x, view.tl.y);
    ::LineTo (view.br.x, view.br.y);
    ::LineTo (view.tl.x, view.br.y);
    ::LineTo (view.tl.x, view.tl.y);

    ::PenPat (&qd.black);
}


/* Proc */ void ViewCookie::HandleDotBox ()
// Cooked
{
    if (everdrawn)
    DrawDotBox (); // First to erase. Always draws at oldxp, oldyp
    oldxp = xp;
    oldyp = yp;
    view.tl.x = oldxp;
    view.tl.y = oldyp;
    view.br.x = oldxp+panw-1;
    view.br.y = oldyp+panh-1;
    DrawDotBox (); // Then to draw
}


// Basic Mouse IO. 4 of 5.
// Cooked
/* Proc */ void ViewCookie::AdjustCursorSelf (Point inPortPt, const EventRecord
&inMacEvent)
{
    xp = inPortPt.h;
    yp = inPortPt.v;

    if (DoDrawBlock)
        HandleDotBox();
}


// Basic Mouse IO. 5 of 5.
// Cooked
/* Proc */ void ViewCookie::SpendTime(const EventRecord& /*inMacEvent*/)
{
}



////////////////////////////////////////////////////////////////////////////
/
//
// Start class WinCookie
//
/* Proc */ WinCookie::WinCookie (MessageT ident,
```

```
                                        Str255 caption,
                                        LWindow    *Caller, // WARNING! Bug of
10/27/97.  This would be the more general
                                                    // LCommander *, but
if so then the procedure screws up the
                                                    // address.
                                        LCommander *inSuperCommander,
                                        GeneralData *gen,
                                        Boolean          ComingFromListener)
                                        : LDialogBox
                                                    (WIND_Block,
                                                    //(WIND_pickerS,
                                                    //(WIND_Tonal,
                                                    // windAttr_Modal +
                                                    windAttr_Regular +
                                                    windAttr_Enabled +
                                                    windAttr_Targetable +
                                                    windAttr_EraseOnUpdate,
                                                    inSuperCommander
                                                    ),
                                        but_cancel__       (0),
                                        but_accept__       (0),
                                        but_revert__       (0)
{
     Int32 groupleft;
     int    initval;
     double initdbl;
     int i;

     capp__  = inSuperCommander;
     gen__ = gen;

     for (i = 0; i <= caption[0]; i++)
          mcaption[i] = caption[i]; // mcaption = caption.

     SPaneInfo   tpi;
     SViewInfo   tvi;
     tvi.imageSize.width      = wwidth;
     tvi.imageSize.height = 800; // 460; // 330
     tvi.scrollPos.h    = 0;
     tvi.scrollPos.v    = 0;
     tvi.scrollUnit.h   = 1;
     tvi.scrollUnit.v   = 1;
     tvi.reconcileOverhang = true;
     tvi.imageSize.width      = wwidth;

     tpi.visible              = true;
     tpi.enabled              = true;
     tpi.bindings.left        = false;
     tpi.bindings.right       = false;
     tpi.bindings.top   = false;
     tpi.bindings.bottom  = false;
     tpi.superView            = this;
     tpi.userCon              = (Int32)capp__; // pass along ptr to App
     tpi.left = tpi.top = 0;
     tpi.width                = wwidth;
     tpi.height               = 800; // 460; // 330
```

```
      secondview__ = new ViewCookie (tpi, tvi, 265, this, capp__, gen__); // 265
(was 274) = yoffset

      //
==========================================================================
==
      initdbl = 0.0;
 }


 #define item break; case


 /* Proc */ void WinCookie::ListenToMessage (MessageT identifier, void *data)
 {
      StringPtr SetMsg;
      StringPtr ClearMsg;

      SDialogResponse   theResponse;
      theResponse.dialogBox = this;
      theResponse.messageParam = data;

      TwoPixelLocs  *pxls;
      int   ri, gi, bi, xi;

      double dvalue;
      Int32  ivalue;
      int      i;

      double maxterm;

      switch (identifier)
      {
            default: break;
      }
 }


 /* Proc */ void WinCookie::ClearPorts()
 {
      secondview__->secondPort = 0;
 }
 // GenTonal.h


 // Destructor (destructor)
 /* Proc */ WinCookie::~WinCookie()
 {
      if (but_cancel__) delete (but_cancel__);
      if (but_accept__) delete (but_accept__);
      if (but_revert__) delete (but_revert__);
 }


 //================================ end of file
====================================
```

```
// ============================================================================
// File: GenCookie.h
// Copyright, 1999, 2000, Xippix, Inc.
//
// ============================================================================

#pragma once


#include <LDialogBox.h>
#include <LPane.h>
#include <LStdControl.h>
#include <LGroupBox.h>

#include "AGAPP.h"

#include "NumScrollBox.h"
#include "DblScrollBox.h"
class GeneralData; // #include "GeneralData.h"


class WinCookie;

////////////////////////////////////////////////////////////////////////////////
//
// Contents
//
// 1. class ViewCookie : public LView, public LBroadcaster, public LPeriodical
// 2. class WinCookie   : public LDialogBox, public LBroadcaster



class ViewCookie : public LView, public LBroadcaster, public LPeriodical
{
public:
     ViewCookie
           (const SPaneInfo &inPaneInfo,
            const SViewInfo &inViewInfo,
            unsigned short yoffset,
            WinCookie   *dlgbox,
            LCommander *inSuperCommander,
            GeneralData *gen);
     // ~ViewCookie();

     typedef struct
     {
           int x,y;
     } uPoint;

     typedef struct
     {
           int status, id, next;
           uPoint tl, br;
     } uRect;

     void RenderRect (uRect *thisrect);
```

```
        void DrawAllBoxes ();
        void HandleDotBox ();
        void DrawDotBox ();

        void DrawSelf ();

        void ClickSelf (const SMouseDownEvent &inMouseDown);
        void EventMouseUp (const EventRecord &inMacEvent);
        void AdjustCursorSelf (Point inPortPt, const EventRecord &inMacEvent);
        // PowerPlant override required for LPeriodical
        virtual void SpendTime(const EventRecord &inMacEvent);

        Point currentPt;
        short xp, yp;
        short oldxp, oldyp;

        unsigned char ItsTall (uRect *r);
        int  FindPredecessor (int block);
        void ReceiveOrderedBoxes();
        void DelRectangle (int block);
        void AddRectangle (int minx, int miny, int maxx, int maxy, int status); //
Add to end

        void FactorBlock (uRect *box);
        unsigned char DoBlocksIntersect (uRect *a, uRect *b, unsigned char
DoCalc);
        int  FindBlockIntersectingBlock (uRect *target);
        void IntersectBox4 (int minx, int miny, int maxx, int maxy);
        void IntersectBox1 (uRect *box);
        int stackcount;

        // The app
        LCommander *capp__;
        GeneralData *gen__;

        WinCookie   *dlg__;

        Rect blonkall;

        unsigned short yoffset__;

        GrafPtr secondPort;  // Watch out

        Rect gframe;
        uRect rlist[480]; // Main rectangle list
        uRect view;
        uRect image;
        uRect intercept;
        int acount; // array count (last position written to).
        int lcount; // linked list count (n positions actually used
        int first;

        uRect slist[480]; // Stack

        unsigned char IncrementRowColumn (int *ri, int *ci);
```

```
      unsigned char bitmap[96][12];
};


class WinCookie : public LDialogBox, public LBroadcaster
{
public:

      WinCookie (MessageT ident,
            Str255 caption,
            LWindow    *Caller, // WARNING! Bug of 10/27/97.  This would be the
more general
                                 // LCommander *, but if so then the procedure
screws up the
                                 // address when called.
            LCommander *inSuperCommander,
             GeneralData *gen,
            Boolean           ComingFromListener);

      ~WinCookie();

      void ClearPorts();
      void ListenToMessage (MessageT inMessage,void *ioParam);

      LCommander   *capp__;
      GeneralData  *gen__;

      ViewCookie   *secondview__;

      LStdCheckBox *cb_brightness__;
      LStdCheckBox *cb_contrast__;

      LStdButton   *but_cancel__;
      LStdButton   *but_accept__;
      LStdButton   *but_revert__;

      Boolean mComingFromListener;

      Str255 mcaption;
};
```